CONDOR - A HUNTER OF IDLE WORKSTATIONS

by

Michael J. Litzkow
Miron Livny
Matt W. Mutka

Computer Sciences Technical Report #730

December 1987

Condor - A Hunter of Idle Workstations

Michael J. Litzkow, Miron Livny, and Matt W. Mutka

Department of Computer Sciences
University of Wisconsin
Madison, WI 53706

ABSTRACT

This paper presents the design, implementation, and performance of the Condor scheduling system. Condor operates in a workstation environment. The system aims to maximize the utilization of workstations with as little interference as possible between the jobs it schedules and the activities of the people who own workstations. It identifies idle workstations and schedules background jobs on them. When the owner of a workstation resumes activity at a station, Condor checkpoints the remote job running on the station and transfers it to another workstation. The system guarantees that the job will eventually complete, and that very little, if any, work will be performed more than once. The system has been operational for more than three months. In this paper we present a performance profile of the system based on data that was accumulated from 23 stations during one month. During the one-month period, nearly 1000 jobs were scheduled by Condor. The system was used by heavy users and light users who consumed approximately 200 CPU days. An analysis of the response times observed by the different users is a clear display of the ability of Condor to protect the rights of light users against heavy users who try to monopolize all free capacity. Since a user of Condor has to devote some local capacity to support the remote execution of his/her jobs, the effectiveness of the remote scheduling system depends on the amount of this capacity. We show that this overhead is very small. On the average, a user has to sacrifice less than one minute of local CPU capacity to acquire a day of remote CPU capacity. Condor has proven to be an extremely effective means to improve the productivity of our computing environment.

1. Introduction

Workstations are powerful machines capable of executing millions of instructions each second. In many environments, individuals are allocated such stations to guarantee fast response to processing demands. In such cases the workstation becomes a private resource of the user who controls access to it. In most cases, the resources of the workstation are under utilized. The processing demands of the owner are much smaller than the capacity of the workstation he/she owns. However, very often some of the users face the problem that the capacity of their workstations is much too small to meet their processing demands. These users would like to take advantage of any available capacity they can access that can support their needs. Modern processing environments that consists of large collections of workstations interconnected by high capacity networks raises the following challenging question: can we satisfy the needs of users who need extra capacity without lowering the quality of service experienced by the owners of under

† This research was supported in part by the National Science Foundation under grants MCS81-05904 and DCR-8512862 and by a Digital Equipment Corporation External Research Grant.

utilized workstations? In other words, can we provide a high quality of service in a highly utilized network of workstations? The Condor scheduling system is our answer to this question. The Condor system schedules long running backgrounds jobs at idle workstations. In this paper we present the design and implementation of Condor and portray its performance. A performance profile based on data accumulated from 23 VAXstation II® workstations over a one-month period is presented along with an analysis of our experience with the usage of the system over the past three months.

A number of researchers have been exploring ways of effectively utilizing computing capacity in networks of workstations [1-8]. This work has been conducted in three areas, which are the analysis of workstation usage patterns, the design of remote capacity allocation algorithms, and the development of remote execution facilities. In the first area of research, workstation usage patterns and their availability as sources of remote execution have been analyzed [1]. An analysis of a group of workstations over 5 months showed that only 30% of their capacity was utilized. The study showed that not only was a large amount of capacity available during the evenings and on weekends, but also during the busiest times of the day. Available intervals were often very long. This makes workstations good candidates for sources of remote processing cycles.

The second area of research is the exploration of algorithms for the management of idle workstation capacity [2]. In a system where long running background jobs are scheduled on idle workstations, it has been observed that some users try to acquire all the capacity available, while others only acquire capacity occasionally. Those who request large amounts of capacity should be granted as much as possible without inhibiting the access to capacity of other users who want small amounts. The *Up-Down* algorithm presented by Mutka and Livny [2] was designed to allow fair access to remote capacity for those who lightly use the system in spite of large demands by heavy users.

The development of remote execution facilities that allow jobs to be executed on idle workstations is the third area of research. A number of papers have reported on the development of systems that allow for remote executions of jobs on idle workstations. These include the NEST project [3], the V-Kernel [4], the Process Server [5], the Remote Unix (RU) facility [6], the process migration facility of Sprite [7], and the Butler [8] system. With the exception of the Remote Unix facility, these systems were not specifically

® VAXstation II is a trademark of Digital Equipment Corporation.

designed to remotely execute long jobs. For example the Butler system does not save the intermediate results of an execution that needs to be moved when a user reclaims the station on which it runs. In this case remote jobs are terminated and all intermediate results are lost. The remote execution facilities of NEST, V-Kernel, Process Server, and Sprite enable job movement during its execution but do not save intermediate results if there is no place to move the job. If a user at a remote site terminates a foreign job running on the station, the foreign job loses all the work it accomplished up to this point. In our department, we use the Remote Unix (RU) facility to execute remote jobs. The RU facility is ideally suited for backgrounds jobs that are computationally intensive and run for long periods without any interaction from users. An unique feature of this facility is *checkpointing*. Checkpointing is the saving of the state of a program during its execution so that it can be restarted at any time, and on any machine in the system. This enables successful completions of jobs that consume months of CPU capacity. When remotely executing programs are stopped due to the shutdown of remote workstations, or when programs are intentionally terminated by remote workstations' owners, programs are resumed from their most recent checkpoints.

This paper presents results that extend previous work with respect to the exploration of effective means of utilizing idle workstation capacity. Previous research of scheduling algorithm design and remote execution facilities are merged into a system where actual user jobs are profiled and the system is measured. The Condor system combines the RU remote execution facility with the Up-Down algorithm for the fair assignment of remote capacity. Our study covers one month in which users' jobs were profiled and the system utilization was monitored. We show the pattern of service demands of users and the quality of service users received.

A new performance measure called *leverage* is introduced. It is a job's ratio of capacity consumed remotely to the capacity consumed locally to support remote execution. When little local capacity is needed to support the execution of remote jobs, the leverage of the jobs is large, When the leverage is small, it is better to execute jobs locally than to consume a great amount of local capacity to support remote executions. We observed the leverage of jobs executing on our system to quantify the benefit the Condor system provided to its users.

Section 2 discusses the design issues of the Condor system and the decisions made to resolve the issues. Included in section 2 are some of the implementation details. Section 3 provides a performance

profile of the system and the impact remote execution has on local workstations. In section 4, we present a discussion of issues that were brought to light due to our implementation. Plans for future work are presented in section 5 and conclusions are given in section 6.

2. System Design

There are over 100 VAXstation II workstations in our department. Since less than 30% of their capacity is utilized [1], a system has been designed and implemented to execute jobs remotely at idle workstations. Within our department there are many users working on problems that need large amounts of computing capacity. A few example problems include studies of load-balancing algorithms [9], simulation of real-time scheduling algorithms [10], studies of neural network learning models [11], and mathematical combinatorial problems [12]. These jobs typically require several hours of CPU time and little interaction with their users. The Condor system is designed to serve these users by executing their long running background jobs at idle workstations. In order to make our system attractive to these users, several issues must be addressed. First, the placement of background jobs should be transparent to users. The system should be responsible for knowing when workstations are idle and users should not need to know where their remote jobs execute. Second, if a remote site running a background job fails, the job should be restarted automatically at some other location to guarantee job completion. Third, since a workstation can serve as a source of remote cycles for others when it is not used by its owner, users expect to receive fair access to cycles when remote capacity is wanted. Fourth, the mechanisms implementing the system are expected to consume very little capacity. Otherwise users would not allow their workstations to be part of a system if it interferes with their local activity.

This paper presents a design and evaluation of a real system that faces these issues. We will describe our remote job execution and recovery facilities, the method of job scheduling, and the system performance. We begin with a description of the structure of the scheduling system.

2.1. Scheduling Structure

The remote job scheduling structure should be transparent to the user. When users have background jobs to run, they should not need to request the remote machines explicitly or know on which machines their jobs are placed. A wide spectrum of scheduling structures could provide this objective. On one end

of the spectrum, a centralized, static coordinator would assign background jobs to execute at available remote workstations. The coordinator would gather system information in order to implement the long-term scheduling policy that the system administrator has chosen. It would know which jobs were waiting and which were executing, and the location of idle stations. At the other end of the spectrum is a distributed approach. The assignment of available processors is accomplished by each workstation cooperating to conduct a scheduling policy. This approach requires negotiations among the workstations to resolve contentions for the available processors.

Both the centralized and the distributed approaches have well known advantages and disadvantages. The centralized approach can efficiently decide which job is next granted a remote processor because each job submitted is registered with the central coordinator. The central location knows both the number of idle workstations and the number of jobs demanding service. The important duties of this location require that it is protected from users so that they do not have direct access to it. Direct access compromises the security of the scheduling policy. A system with a static central coordinator that keeps all jobs' state and workstation availability information is not easily extendible and is critically subject to failure. If the central coordinator fails, all scheduling in the system would cease. In the distributed scheduling system, each requesting workstation does its own searching for idle workstations. Message exchanges among contending workstations would be required to place jobs at idle workstations. This is less efficient than a centralized scheme when deciding which job should be next allocated a processor. However, the distributed scheduling approach is not subject to failure if a single station quits operating.

We have decided to follow an approach for structuring the background job scheduler that lies between a centralized, static approach and the fully distributed approach. This approach uses the efficiency of scheduling with a central node to avoid the overhead of messages to decide which workstations should be allocated available capacity. The jobs' state information is kept at individual workstations. Each workstation has the responsibility of scheduling its own jobs. A workstation decides which jobs in its queue have the highest priority. The central coordinator merely assigns capacity to workstations which they use to schedule their own jobs.

Figure 1 illustrates our approach to structuring the system scheduling. On each workstation is a local scheduler and a background job queue. The jobs that the user submits are placed in the background job

queue. One workstation holds the central coordinator in addition to a local scheduler and background job queue. In our implementation, every two minutes the central coordinator polls the stations to see which stations are available to serve as sources for remote cycles, and which stations have background jobs waiting. Between successive polls, each local scheduler monitors its station to see if it is available as a source of remote capacity. If a background job is running on the workstation, the local scheduler checks every ½ minute to see if the background job should be preempted because the local user has resumed using the station. If a user has resumed using the station, the local scheduler will immediately preempt the background job so that the user can have the workstation's capacity under his control. The central coordinator allocates capacity from idle workstations to local schedulers on workstations that have background jobs waiting. A local scheduler with more than one background job waiting makes its own decision of which job should be executed next.

Our structure follows the principle that workstations are autonomous computing resources and they should be managed by their own users. This also helps to keep the responsibilities of the coordinator simple. Simplicity is important so that a central site is not required to maintain a great amount of information about each workstation. This allows the system to be extendible to a large number of workstations and

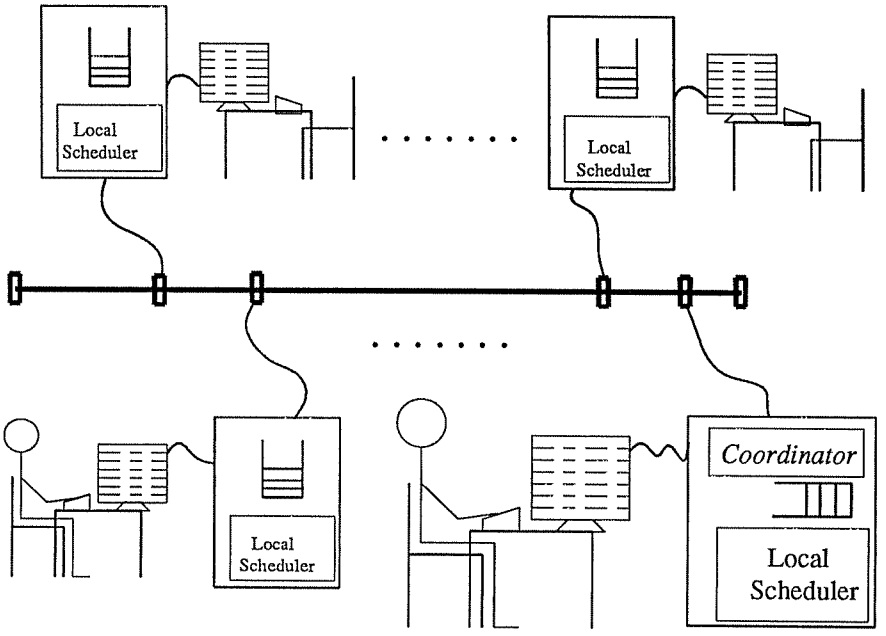

Figure 1: The Condor Scheduling Structure.

eases the required recovery when the centralized coordinator fails. Local schedulers are not affected if a remote site discontinues service. If the site on which the coordinator is executing fails, remotely executing jobs initiated and executing on other machines are not affected. Only the allocation of new capacity to requesting users is affected. Since the coordinator has few duties, its recovery at another site is simplified in relation to a fully centralized strategy. In order to balance the burden of coordination, the central coordinator can be moved to other locations. However, we have observed that the coordinator contributes less than 1% to the CPU consumption of a workstation so that there is probably little need to move the coordinator.

In order to schedule jobs remotely, a remote execution facility is needed. Since our workstations operate under the Berkeley BSD 4.3 Unix® operating system, we decided to have a remote execution facility that is compatible with our local job execution facility. This led to the development of the *Remote Unix* (RU) facility [8].

2.2. The Remote Unix (RU) Facility

Remote Unix turns idle workstations into cycle servers. When RU is explicitly invoked, a *shadow* process runs locally as the surrogate of the process running on the remote machine. Any Unix system calls of a program on the remote machine invokes a library routine which communicates with the shadow process. A message indicating the type of system call is sent to the shadow process on the local machine.

When someone resumes using a workstation that is executing a remote job, the job must be stopped. If the state of the stopped job is not preserved, as is the case in the Butler system [7], all the work accomplished by the job is lost. Because background jobs can require several hours of capacity, it is important that the system restart background jobs without losing all the work previously accomplished. In the Condor system, the intermediate state from which background jobs can be restarted is made possible by a checkpointing feature of the RU facility.

® Unix is a trademark of AT&T Bell Laboratories.

2.3. Checkpointing

The RU facility checkpoints jobs when they are removed from remote locations. The checkpointing of a program is the saving of the state of the program so that its execution can be restarted. The state of an RU program is the text, data, bss, and the stack segments of the program, the registers, the status of open files, and any messages sent by the program to its shadow for which a reply has not been received. In our system, we do not need to save messages since checkpointing is deferred until the shadow's reply has been received. The text of the program contains the executable code, the data segment contains the initialized variables of the program, and the bss segment holds the uninitialized variables. Because it is assumed that there is no self-modifying code in the program, the text segment remains unchanged during the execution of the program. Therefore the text segment is expected not to be essential in a checkpoint file. However, programs can execute for a very long time, perhaps months. A user might want to modify a program that has its executable file running as an RU job. For this reason, we save the text segment. Otherwise, the user would have to make sure that the new program's executable file is given a new name when there is an old version running.

2.4. Fair Access to Remote Cycles

Once a scheduling structure has been established, we need to understand the characteristics of the users in order to design algorithms that meet their needs. We have observed that some users try to consume all available capacity for long periods. Other light users only consume remote cycles occasionally. In order to serve all users fairly, we need to take into account their workload. Otherwise, heavy users might inhibit light users' access to remote cycles.

To provide fair access to resources, we manage available capacity with the Up-Down algorithm [2]. This algorithm enables heavy users to maintain steady access to remote cycles while maintaining fair access to cycles for light users. It trades off the remote cycles users have received with the time they have waited to receive them. The scheduling coordinator does this by maintaining a schedule index for each workstation. When remote capacity is allocated to a workstation, the index is increased. When a workstation wants remote capacity, but is denied access to it, the index is decreased. The priority to remote cycles of a workstation is determined by the value of its index. Initially the index for each station is zero. The

indexes of the workstations are updated periodically. Every two minutes the coordinator will check if any stations have new jobs to execute. If a station with higher priority has a job to execute, and there are no idle stations, the coordinator preempts a remotely executing job from a station with lower priority. After the preempted job is checkpointed, the newly available capacity will be assigned to the high priority station. Further details of the algorithm and an evaluation of its performance is given by Mutka and Livny[2].

The implementation of the system has given us an opportunity to measure its performance under actual usage. This enables measurement of the costs and benefits of providing a background scheduling service. The next section discusses the detailed measurements we obtained from the system when it was used by members of our department.

3. Performance

The performance results we report are from preliminary observations of the Condor system. We present details of the way the system was used and analyze the quality of service it provided. This analysis includes the wait ratios users endure when they submit background jobs and the cost suffered by users at their local workstation to support remotely executing jobs. Our results are based on observing 23 workstations for 1 month. Table 1 summarizes the activity of users during the month. It presents the number of jobs each user submitted, and the average service demand of a job per user. User A accounted for most of the consumption of remote capacity. This *heavy* user often tried to execute as many remote jobs as there were workstations in the system. The other users of Condor consumed capacity occasionally and can be classified as *light* users.

The service demand of jobs submitted to the system were typically several hours in length. With the exception of User D, all users had an expected demand per job that was greater than 1 hour. Figure 2

User	Number of Jobs	Percentage of Total Jobs	Average Demand/Job (in Hours)	Total Demand (in Hours)	Percentage of Total Demand
A	690	75	6.2	4278	90
B	138	15	2.5	345	7
C	39	4	2.6	101	2
D	40	4	0.7	28	0.6
E	11	1	1.7	19	0.4
Total	918	100	5.2	4771	100

Table 1: Profile of User Service Requests.

shows the cumulative frequency distribution of jobs served by the system. For each hour i, the curve shows the percentage of jobs whose service demand was less than i hours. The average service demand was $\approx$ 5 hours. The median service demand was less than 3 hours because shorter jobs were submitted more frequently than longer jobs.

Jobs arrived at the system in batches. Figure 3 depicts the queue length of jobs in the system on an hourly basis. The dotted line represents the queue length of light users. Jobs in service are considered part of the queue. The difference between the total and light users' queue lengths is the heavy user's queue length. The figure shows that the heavy user kept more than 30 jobs in the system for long periods. The light users submitted their jobs in batches of $\approx$ 5 jobs.

We evaluated the quality of service users receive for the remote execution of their jobs. One measure of the quality of service is the wait ratio of jobs submitted for remote execution. The wait ratio of a job is the ratio between the amount of time a job waits for service and its service time. The average of observed wait ratios is illustrated in Figure 4. The solid line is the average wait ratio of all jobs. The dashed line is the wait ratio of the light users. The figure shows that in most cases light users did not wait at all. The average wait ratio results are dominated by the wait ratio of the heavy user. The heavy user

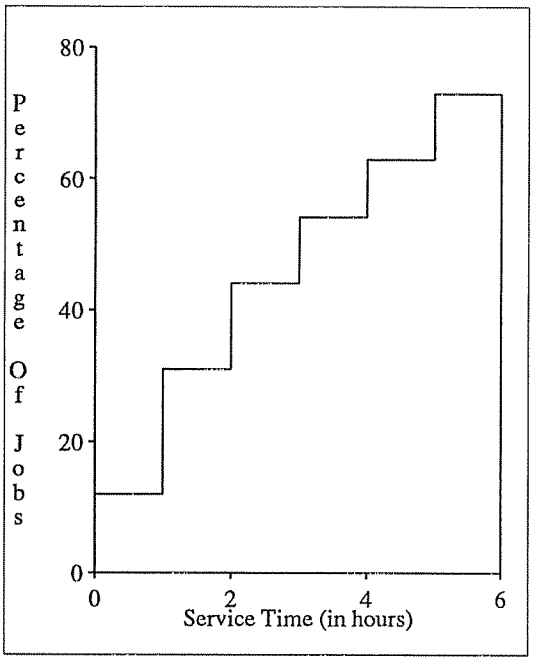

Figure 2: Profile Of Service Demand.

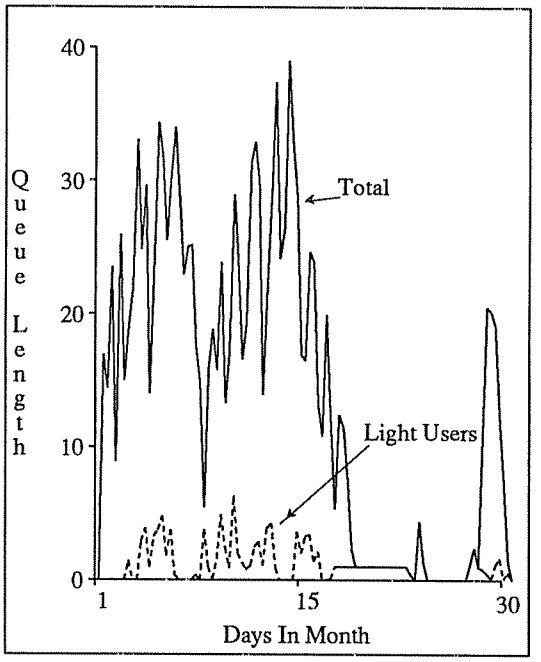

Figure 3: Queue Length.

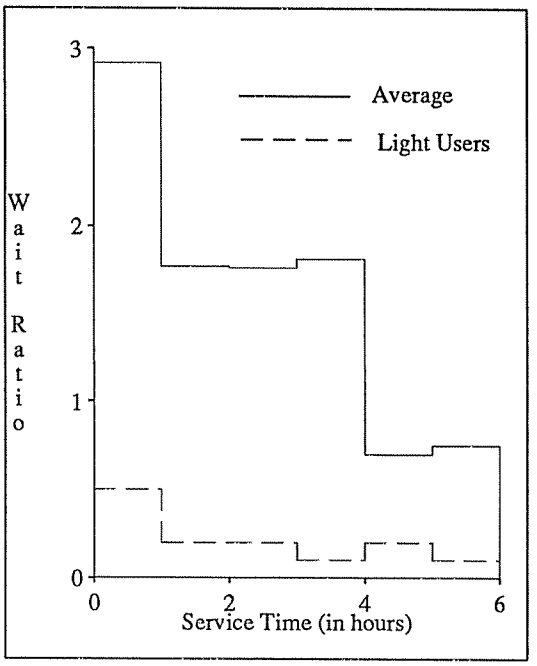

Figure 4: Average Wait Ratio.

waited significantly more for his jobs. This is due to the Up-Down algorithm giving steady access to light users without allowing heavy users to dominate the system. The light users obtained remote resources regardless of whether the heavy user increased or decreased his load. Requests of the light users were typically small enough that available capacity could be immediately allocated to them. The Up-Down algorithm allocated remote capacity to light users and preempted the heavy user. When the light users' jobs were completed, the heavy user's jobs were resumed to consume available capacity. Typically the heavy user was allocated some capacity since the light users' requests were not large enough to consume all available capacity.

We measured the amount of extra capacity the 23 workstations provided to Condor users. During the observed period, 12438 hours were available for remote execution, of which 4771 machine hours of capacity was consumed by the Condor system. Note that almost 200 machine days of capacity that otherwise would have been lost were consumed by the Condor system! Figure 5 shows how the utilization varied over time. The solid line is the system utilization which is the combination of local activity and remote executions. The dashed line shows the local workstation utilization. The local activity remained low for the month period. The average local utilization for the month was 25%. The solid line shows that

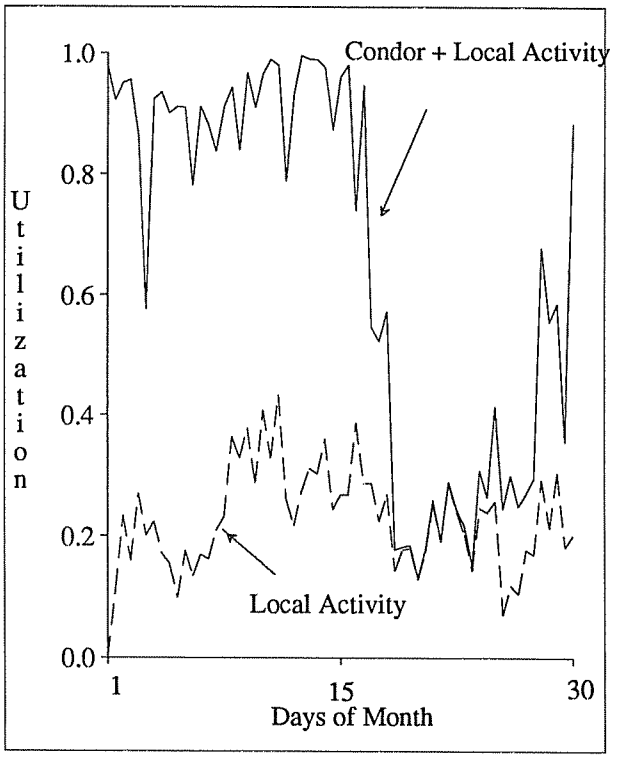

Figure 5: Utilization of Remote Resources.

often all workstations were utilized. The Condor system identified available capacity and allocated it to its users.

Each day of the month the amount of available capacity in the system varied. Figure 6 gives a closer view of the utilization of the system over one working week (Monday through Friday). Notice the peaks of local activity during the day, and how the capacity decreased in the evenings. The range of utilization generally varied from 20% in the evenings and nights to 50% for short peak periods in the afternoons. The entire capacity of the 23 workstations was utilized for long intervals. The queue length of light users and the total queue length for that week is given in Figure 7. Notice the sharp rises in the queue length which represents batch arrival of jobs. Much of the time during the week the queue length of the heavy user was larger than the number of machines available.

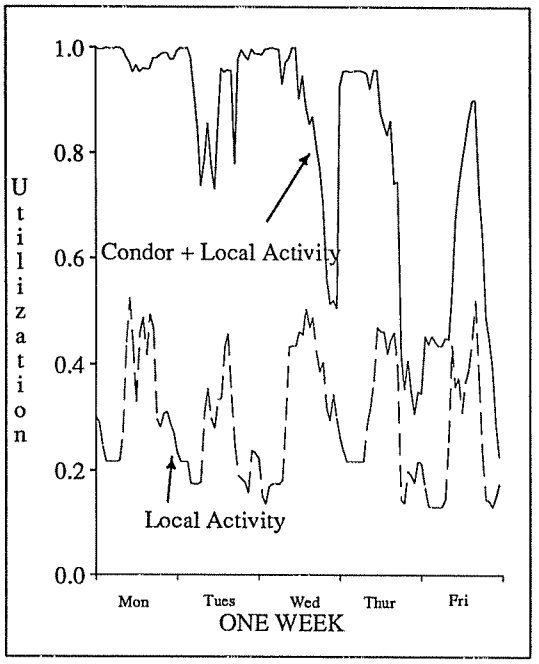

Figure 6: Utilization for One Week.

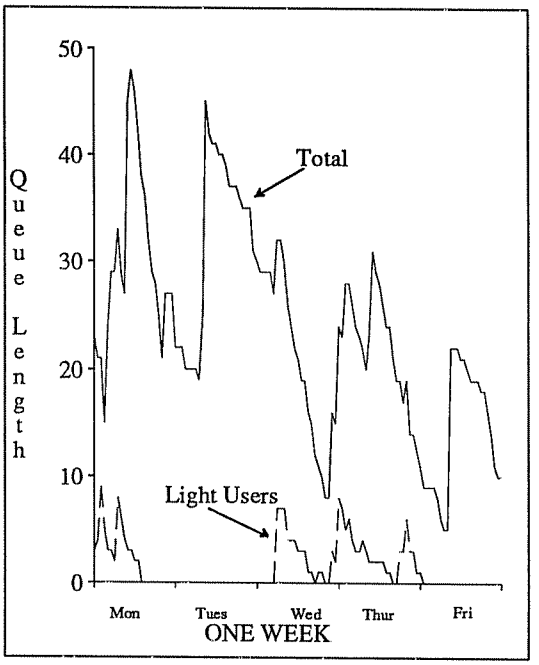

Figure 7: Queue Lengths for One Week.

3.1. Impact on Local Workstations

The implementation of remote execution facilities should be efficient so that users at workstations need not use much of their local capacity to support remote executions. We studied the impact the remote execution facility has on users at their workstations. A user has to devote some local capacity to support the placement and checkpointing of remote jobs and the execution of system calls. In addition, a local scheduler and the coordinator consume some resources.

It is important to keep the capacity consumed by the coordinator and each local scheduler small since some users might rarely use the remote execution facility. Our observations show that these cost are indeed small. The local scheduler of a station with background jobs running has been observed to consume less than 1% of a station's capacity. This capacity is independent of the size of the system. The consumption of capacity by the coordinator has been observed to be less than 1% of a workstation's capacity as well. The size of the system is expected to affect the amount of capacity consumed by the coordinator. We have observed a system with as many as 40 workstations. Even with this system size, the coordinator consumes less than 1%. This leads us to believe that a coordinator can manage as many as 100 workstations with only a small impact on the workstation that hosts it.

We measured the costs that remotely executing jobs bring on the workstations from which they were submitted. To support the remote execution of background jobs, the local workstation has to transfer jobs to remote sites, checkpoint them when they are preempted, and execute their system calls. This support can have a significant impact on the local workstation. The costs associated with this support depend on the the *costs* and *rates* of *placing* and *checkpointing* remote jobs, and the *costs* and *rates* of executing *system calls.*

The capacity required to place and checkpoint a remote job depends on the size of the job. Placing and checkpointing jobs consume approximately 5 seconds per megabyte of the checkpoint file. We observed that the average checkpoint file size was ½ megabyte. This means that the average cost of placement and checkpointing was approximately 2½ seconds.

The rate at which jobs were checkpointed after they were initially placed is shown in Figure 8. This rate is the number of times per hour that a remotely executing job is moved from one location to another. Jobs are checkpointed when the location at which they have been running becomes unavailable for remote execution. In addition, jobs can be checkpointed when the coordinator decides that one user requesting remote cycles has priority over another user. The rate of checkpointing was relatively steady over the range of service demands, with the exception of the short jobs. The reason that longer jobs have a lower rate of checkpointing can be explain in terms of the local usage patterns of workstations. When jobs are preempted due to local user activity, they will be placed at another remote location if one is available. Since local workstation activity is not uniform across the system, some workstations tend to be available for long periods, and other workstations tend to be available for much shorter periods [1]. Long jobs have a lower checkpoint rate because eventually they are placed at a workstation that experiences no local activity.

System calls by a remotely executing job can have a significant impact on a local workstation. The average capacity consumed on a VAXstation II to support a remote job executing a system call is approximately 10 msec. If a job is executing locally, the system call consumes 1/20 the capacity of the remote system call. Programs executing large numbers of system calls, such as reads or writes, in proportion to other instructions would be better if they were executed locally instead of remotely. For a remotely executing job with an extreme number of system calls, a local workstation supporting the remote system calls would

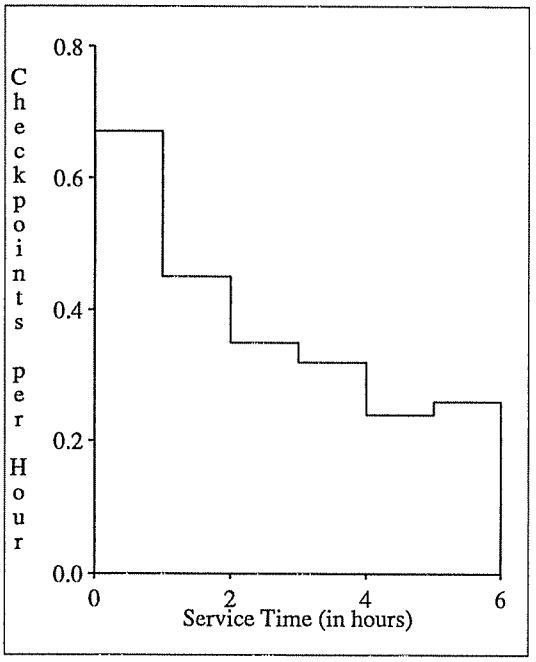

Figure 8: Rate Of Checkpointing.

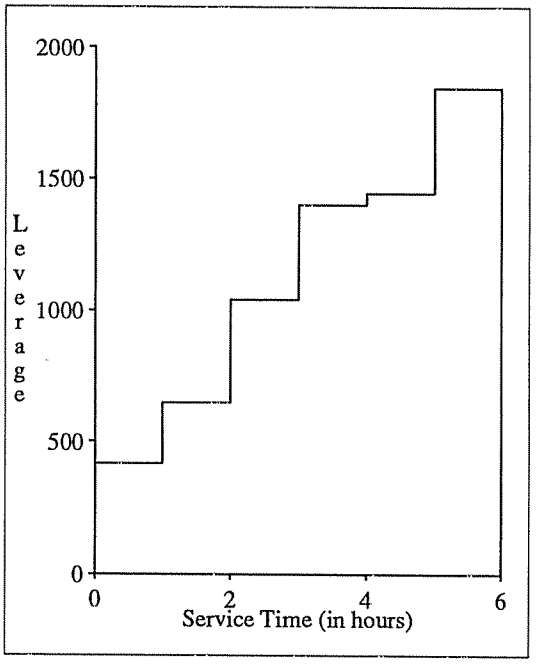

Figure 9: Remote Execution Leverage.

consume more capacity than the amount of useful work accomplished at the remote site.

We define a new performance measure called *leverage* to compare the amount of effort a local workstation must endure to benefit from having useful work conducted remotely. The leverage of a remote job is defined as the amount of remote capacity consumed to execute a job divided by the amount of local capacity consumed to support remote execution. The local capacity is the combination of capacity used to support placement, checkpointing, and system calls. If more capacity is consumed locally to support remote executions than what is actually accomplished remotely, the leverage of the job is less than 1. Figure 9 shows a profile of the leverage of jobs. The average leverage was approximately 1300. This means for every 1 minute of local capacity consumed to support remote execution of jobs, nearly 22 hours of remote capacity was received by the users! Longer jobs had a larger leverage than shorter jobs. This is on account of the rate of checkpointing for shorter jobs was higher than for longer jobs, and the amount of input/output for the shorter jobs was relatively the same as the input/output of longer jobs. Nevertheless, the leverage for jobs with service demands less than 2 hours averaged approximately 600. This means that even a short job with only a service demand of 1 hour required only about 6 seconds of local capacity to support remote execution.

4. Discussion

The implementation of the Condor system brought a clearer understanding of several issues. Many of these issues relate to the nature of background jobs and the large amount of memory needed for their remote execution. For example, if a job is to be executed remotely, it must be placed on the remote station's disk. Because users of workstations often do little to manage their own disk space, users let their disk become full. When a disk is full, a remote job cannot be placed on the workstation for remote execution. Even if a workstation is idle so that its processor is available for executing remote jobs, the disk might be full so that no remote job can execute there. The coordinator must know not only which workstation's processor is available, but must know how much disk space is available on each workstation.

The issue of disk space affects users in another way. Users often like to execute many background jobs at a time. If users do not have much local disk available, they will be restricted on the number of background jobs that can be executing simultaneously. The restriction occurs since checkpoint files of remotely executing background jobs are kept locally. Space can be saved if disk servers from additional hardware are implemented to store checkpoint files of jobs. Another solution to the disk space problem is to share text segments of programs. This is effective since users often submit several occurrences of the same job to the system with only different parameters to evaluate. An example is when users submit simulation programs to the system. Only one copy of the text segment might be needed for several job executions.

Because placing and checkpointing remote jobs has an impact on a local workstation and the network, our implementation does not try to place or checkpoint several jobs simultaneously. We have noticed that if several machines are available, and users have several background jobs waiting for service, the performance of the local machine is severely degraded if all jobs are placed at the same time. Our implementation places a single job remotely every two minutes to distribute over time the impact on local workstations and the network.

Our design philosophy has been to ensure that the Condor system does not interfere with users and their local activity. Remote jobs are only executed when there is no local activity. However, one element of our implementation differs with our design philosophy. When local activity resumes at a workstation where a foreign job is running, the foreign job is stopped on the station and is kept there to see if the

workstation will soon be available. If it does not become available within 5 minutes, the job will be checkpointed and moved from the location. The strategy has worked well since many of the workstations' unavailable intervals are short. However, it does not completely follow a model where users reclaim all local resources as soon as they return to their workstations. The CPUs are immediately returned, but disk space consumed by remote jobs is not released until the checkpoint files are moved. If a user has little local available disk space, the checkpoint file might interfere with local activity until the file is moved. We are considering modifying our strategy so that checkpoints of remote executions are periodically taken. When a workstation's owner resumes activity at a location executing a remote job, the new strategy is to kill the job immediately. This minimizes any interference a remote job has with the owner of a workstation. The only work lost is that between the job's most recent checkpoint and the time it was terminated.

5. Further Work

Our work is the first step in exploring design and implementation issues regarding background job scheduling in a network of workstations. There are several performance evaluation and implementation issues which we intend to further study. Some of the example issues are:

(1) Other work [1] has found that workstations with long available intervals tend to have their next available interval long. Workstations with short available intervals tend to have their next available intervals short. This correlation means that the coordinator could choose sources of remote cycles on the basis of the history of workstation availability . We intend to study the impact on the number of preemptions long running jobs suffer when we use knowledge of past available interval lengths.

(2) We are considering the implementation of the unix system calls *fork(2)*, *exec(2)*, and *pipe(2)* [13] to allow parallel programs to be executed on the system. This facility would introduce many scheduling problems.

(3) The implementation of a reservation system would improve the computing service available to users. Reservations guarantee computing capacity for users in advance in order to conduct experiments in distributed computations. Many important issues are open on how to manage a reservation system in which workstations become available whenever their owners are not using them.

(4) We are considering porting our system to the SUN [14] workstations. This system means that a background job compiled into two different binary files could be executed at either a VAXstation II or SUN workstation. This system leads to interesting scheduling questions regarding at which workstation should a job be placed. The decision of placement should take into account the usage patterns of each type of workstation. Once a job has been placed on one type of workstation, the job could not be moved to the other type of workstation without losing all the work done on the first type of workstation.

6. Conclusions

Networks with workstations have increased in great numbers in recent years. These networks represent powerful computing environments that were previously only available to users at institutions with supercomputers. With the implementation of the Condor system, users can expand their capacity to that of the entire computing network. This paper discusses a system that effectively utilizes idle workstation capacity and presents a profile of its performance. The results are from a one-month observation of the system where actual users obtained capacity from workstations that otherwise would have been idle.

Condor has proven to be an extremely effective means of improving the productivity of our computing environment. For a system of 23 workstations, large amounts of capacity were observed to be available for remote execution. About 75% of the time the workstations were available as sources of remote cycles. Our system caused the workstations to be fully utilized for long periods. Over a one-month period, users consumed as much as 200 machine days of computing cycles from available workstations. The checkpointing feature of our remote execution facility insured users that their jobs would complete regardless if their jobs were forced by users at remote locations to stop, or if remote locations failed. We showed that users need only to dedicate an extremely small amount of workstation capacity locally to received huge amounts of remote cycles. We report that the leverage of remote execution observed was 1300, which means for every minute of local capacity supplied, almost 1 day of remote CPU capacity was received.

Acknowledgements

We would like to thank Don Nuehengen and Tom Virgilio for their pioneering work on the remote system call implementation.

References

[1] M. W. Mutka and M. Livny, "Profiling Workstations' Available Capacity for Remote Execution," *Performance '87, Proceedings of the 12th IFIP WG 7.3 Symposium on Computer Performance*, Brussels, Belgium, (December 7-9, 1987).

[2] M. W. Mutka and M. Livny, "Scheduling Remote Processing Capacity in a Workstation-Processor Bank Computing System", *Proceedings of the 7th International Conference of Distributed Computing Systems*, Berlin, West Germany, pp. 2-9, (September 21-25, 1987).

[3] R. Agrawal and A. K. Ezzat, "Processor Sharing In Nest: A Network Of Computer Workstations," *Proceedings of 1st International Conference on Computer Workstations*, (November, 1985).

[4] M. M. Theimer, K. A. Lantz, and D. R. Cheriton, "Preemptable Remote Execution Facilities for the V-System," *Proceedings of the 10th Symp. on Operating Systems Principles*, pp. 2-12, (December, 1985).

[5] R. Hagmann, "Processor Server: Sharing Processing Power in a Workstation Environment," *Proceedings of the 6th IEEE Distributed Computing Conference*, Cambridge, MA, (May, 1986), pp. 260-267.

[6] M. Litzkow, "Remote Unix," Proceedings of 1987 Summer Usenix Conferences, Phoenix, Arizona, (June, 1987).

[7] F. Douglis and J. Ousterhout, "Process Migration in the Sprite Operating System," *Proceedings of the 7th International Conference of Distributed Computing Systems*, Berlin, West Germany, pp. 18-25, (September 21-25, 1987).

[8] D. A. Nichols, "Using Idle Workstations in a Shared Computing Environment", *Proceedings of the 11th Symp. on Operating System Principles*, pp.5-12, (November, 1987).

[9] P. Krueger and M. Livny, "The Diverse Objectives of Distributed Scheduling Policies", *Proceedings of the 7th International Conference of Distributed Computing Systems*, Berlin, West Germany, pp. 242-249, (September 21-25, 1987).

[10] H.-Y. Chang and M. Livny, "Priority in Distributed Systems," *Proceedings of the Real-Time Systems Symposium, (December, 1985).*

[11] P. Sandon, "Learning Object-Centered Representations," Ph. D. Thesis, University of Wisconsin, Madison, Wisconsin, (August, 1987).

[12] D. Chavey, *Private Correspondence*, University of Wisconsin, Madison, Wisconsin, (December, 1986).

[13] *Unix 4.3BSD System Call Manual.*

[14] A. Bachtolsheim, V. R. Pratt, and F. Baskett, "The SUN Workstation Architecture," Technical Report 229, Computer Sciences Laboratory, Stanford University, Palo Alto, California, (February, 1982).